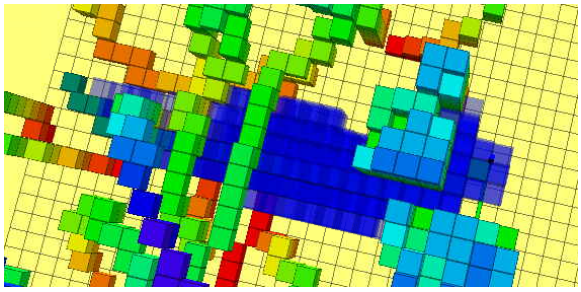
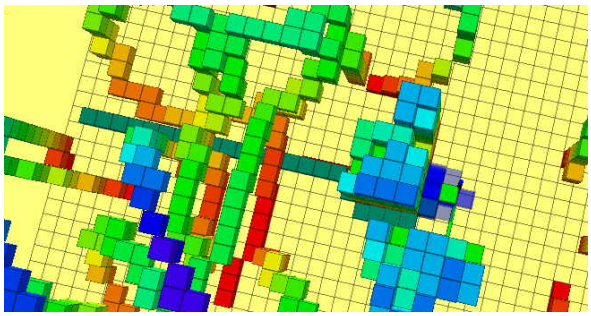
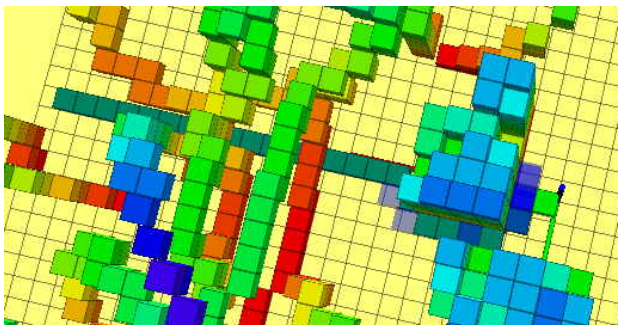
1. getHeu加入了计算不同距离的方程
2. 没有加任何代码
3. 将openSet.begin()取出用currentPtr指向该节点，因为multimap已经将键排序，begin就是目前最优的节点。
4. 将currentPtr的上下左右前后26个节点遍历。将所有在Map内，没有障碍物且不在closedSet里的节点加入neighborPtrSets，并且将currentPtr指向的与加入的节点的距离算出，并且加入edgeCostSets
5. 取出第i个节点，并且提前计算节点的gScore已经fScore，为了之后比较fScore哪个更优。
6. 如果加入的节点未探索过，先用当前节点的gScore加上当前节点对应的edgeCostSets里的cost算出新节点的gScore。再计算新节点的Heuristic，以及fScore和camefrom指向当前节点。将新的节点的fScore作为键，加入openSet
7. 如果加入的节点已经在openSet, 我们先检查新节点的gScore是否比之前更低，如果否，continue，如果是，将新节点的camefrom设为当前节点，并且更新fScore，gScore
8. 从terminatePtr不断更新当前指针为上一次指针的camefrom开始直到gScore==0，将所有访问过的指针加入gridPath
Tie breaker我加入了cross product。并且overload了getHeu加了一个参数startPtr
我在实现时改了isOccupied()，如果节点在地图之外，也将返回True。



L2 Norm

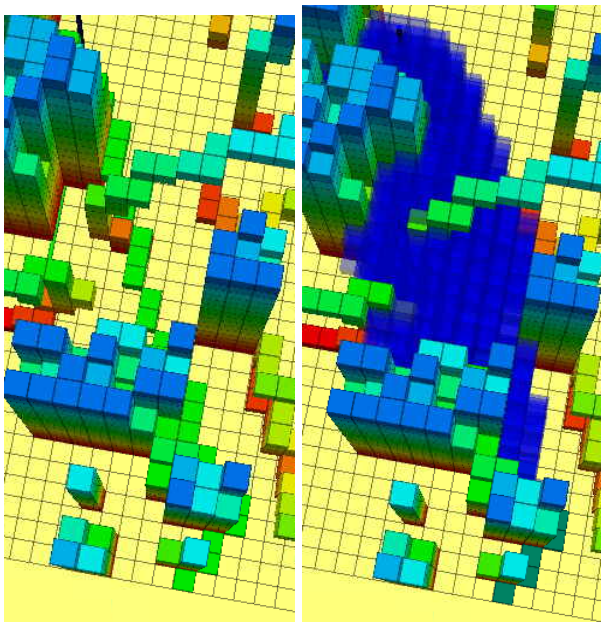


L1 Norm

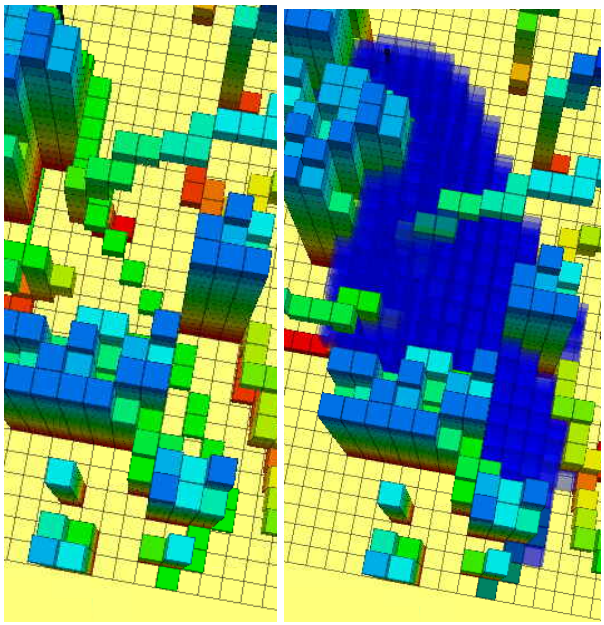


对角线距离

可以看出在这个情况下，L2 norm的A*是最耗算力的，L1和对角线距离在这里表现比较好，这是由于这两个Heuristic和最优解的cost最接近。



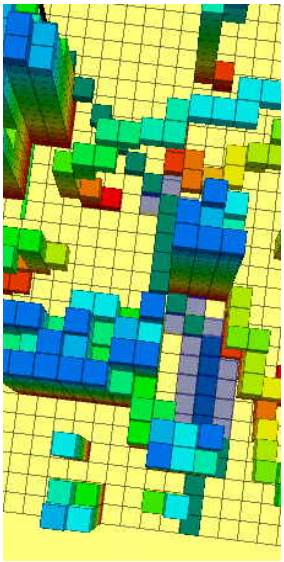
L2 Norm with tie breaker



Normal L2 Norm

可以看出在这个例子里，添加tie breaker优于不加tie breaker，但是并不是数量级的区别。普通L2一共访问了1316个节点，添加tie breaker之后访问了915个节点。tie breaker主要避免了访问非常外围的点，一共用了0.775ms，普通L2一共用了1.27ms。

这里没有数量级上的区别主要是由于最优路径上的障碍物很多，导致了heuristic与最优cost相差较大导致的。



L1 norm

看上去L1 norm总是快于其他算法，我认为这是由于L1 norm并不符合admissible的标准，只有对L1 norm乘上一个系数才能保证L1 norm的heuristic永远小于最优解。 纯L1 norm的heuristic可以很快算出答案，但是给出的解并不能保证最优。